COMPUTER ENGINEERING SERIES

Fundamentals of Software Testing

2nd Edition Revised and Updated

Bernard Homès

ISTE WILEY

Fundamentals of Software Testing

Revised and Updated 2nd Edition

Fundamentals of Software Testing

Bernard Homès

ISTE

WILEY

First edition published 2011 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc., © ISTE Ltd 2011.

This edition published 2024 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Control Number: 2024932622

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-78630-982-2

Contents

Preface xi

Glossary xvii

Chapter 1. Fundamentals of Testing 1

1.1. What is testing? 1
1.1.1. Software and systems context 1
1.1.2. Causes of software defects 3
1.1.3. Role of testing in software development, maintenance and operations 5
1.1.4. Tests and quality 5
1.1.5. Terminology 7
1.2. What is testing? 8
1.2.1. Origin of defects 8
1.2.2. Common goals of testing 9
1.2.3. Examples of objectives for testing 9
1.2.4. Test and debugging 10
1.3. Paradoxes and main principles 11
1.3.1. Testing identifies the presence of defects 11
1.3.2. Exhaustive testing is impossible 11
1.3.3. Early testing 12
1.3.4. Defect clustering 12
1.3.5. Pesticide paradox 13
1.3.6. Testing is context dependent 13
1.3.7. Absence-of-errors fallacy 14
1.4. Test activities, testware and test roles 14
1.4.1. Planning 15
1.4.2. Monitoring and control 16
1.4.3. Test analysis and design 17

1.4.4. Test implementation 20
1.4.5. Test execution 21
1.4.6. Reporting 23
1.4.7. Test completion activities 24
1.4.8. The value of traceability 25
1.4.9. Impact of context on the test process 25
1.5. Roles in testing 26
1.6. Essential skills and “good practices” in testing 26
1.6.1. Generic skills 26
1.6.2. Specific skills 27
1.6.3. Whole team approach 27
1.6.4. Independence of testing 28
1.6.5. Levels of independence 29
1.6.6. Adapt to objectives 30
1.6.7. Destructive or constructive? 32
1.6.8. People skills 32
1.6.9. Change of perspective 33
1.7. Testers and code of ethics (FL 1.6) 33
1.7.1. Public 34
1.7.2. Client and employer 35
1.7.3. Product 35
1.7.4. Judgment 37
1.7.5. Management 37
1.7.6. Profession 38
1.7.7. Colleagues 39
1.7.8. Self 40
1.8. Sample exam questions 41

Chapter 2. Testing Throughout the Software Life Cycle 47

2.1. Testing through the software development life cycle 47
2.1.1. Sequential models 48
2.1.2. Iterative models 50
2.1.3. Incremental model 52
2.1.4. RAD 54
2.1.5. Agile models 55
2.1.6. Selection of a development model 61
2.1.7. Positioning tests 62
2.1.8. Test-first and shift-left approaches 63
2.2. Test levels and test types 64
2.2.1. Component-level testing or component tests 65
2.2.2. Integration-level testing or integration tests 66
2.2.3. System tests 68

2.2.4. Acceptance tests. 70
2.2.5. Other levels 72
2.3. Types of tests. 72
2.3.1. Functional tests 73
2.3.2. Nonfunctional tests 74
2.3.3. Tests based on the structure or architecture of the software. 76
2.3.4. Tests associated with changes 77
2.3.5. Comparisons and examples 79
2.4. Test and maintenance 80
2.4.1. Maintenance context 81
2.4.2. Evolutive maintenance 81
2.4.3. Corrective maintenance 82
2.4.4. Retirement and replacement 83
2.4.5. Regression test policies. 83
2.4.6. SLA validation and acceptance 86
2.5. Oracles 86
2.5.1. Problems with oracles 87
2.5.2. Sources of oracles. 87
2.5.3. Oracle usage. 88
2.6. Process improvements. 89
2.6.1. Objectives 89
2.6.2. Measurements 89
2.6.3. Retrospectives and improvements. 89
2.7. Specific cases 90
2.7.1. Performance tests 90
2.7.2. Maintainability tests 91
2.8. Sample exam questions 91

Chapter 3. Static Testing 97

3.1. Static techniques and the test process 97
3.2. Review process 100
3.2.1. Types of reviews 101
3.2.2. Roles and responsibilities during reviews. 106
3.2.3. Phases of reviews 109
3.2.4. Success factors for reviews 122
3.2.5. Comparison of the types of reviews. 123
3.3. Static analysis by tools 125
3.3.1. Types of static analysis. 125
3.3.2. Types of defects that can be identified 130
3.3.3. Data flow analysis 131
3.4. Added value of static activities 136
3.5. Sample exam questions 137

Chapter 4. Test Design Techniques 141

4.1. The test development process. 143
4.1.1. Terminology. 143
4.1.2. Traceability 144
4.2. Categories of test design techniques 146
4.2.1. Black box, white box or gray box 147
4.2.2. Experience-based techniques 148
4.2.3. Test characteristics 149
4.2.4. Limitations and assumptions. 149
4.3. Black-box techniques 151
4.3.1. Equivalence partitioning 152
4.3.2. Boundary value analysis 158
4.3.3. Decision tables 161
4.3.4. Other combinational techniques 166
4.3.5. State transition testing 166
4.3.6. Use case testing 173
4.3.7. Limitations and assumptions. 175
4.4. Structure-based techniques 175
4.4.1. Statement testing and coverage 178
4.4.2. Decision testing and coverage 183
4.4.3. Other structure-based techniques 186
4.4.4. MC/DC coverage 188
4.4.5. Limitations and assumptions of structure-based testing 189
4.4.6. Coverage level and exit criteria 190
4.5. Experience-based technique 190
4.5.1. Attacks. 191
4.5.2. Defect taxonomies 192
4.5.3. Error guessing and ad hoc testing 193
4.5.4. Exploratory testing 194
4.5.5. Limitations and assumptions. 195
4.6. Collaboration-based test approaches 196
4.6.1. Collaborative user stories 196
4.6.2. Acceptance criteria 197
4.6.3. Acceptance test-driven development 197
4.7. Choosing test techniques 198
4.8. Sample exam questions 200

Chapter 5. Test Management 209

5.1. Test organization. 209
5.1.1. Independence levels 209
5.1.2. Roles and responsibilities 212
5.1.3. Human and contractual aspects 214

5.2. Test planning and estimation . . . 215
5.2.1. Planning and evaluation activities . . . 218
5.2.2. Test planning activities . . . 229
5.2.3. Test documentation . . . 231
5.2.4. Entry and exit criteria for test activities . . . 236
5.3. Test progress monitoring and control (FL 5.3) . . . 240
5.4. Reporting . . . 243
5.4.1. What to report, to whom and how? . . . 243
5.4.2. Statistics and graphs . . . 245
5.5. Transverse processes and activities . . . 248
5.5.1. Test data definition . . . 248
5.5.2. Configuration management (FL 5.4) . . . 249
5.5.3. Change management . . . 250
5.6. Risk management (FL 5.2) . . . 250
5.6.1. Principles of risk management . . . 251
5.6.2. Project risks and product risks . . . 255
5.6.3. Introduction to risk management . . . 256
5.7. Defect management (FL 5.5) . . . 259
5.7.1. Introduction to defect management . . . 260
5.7.2. Defect identification . . . 261
5.7.3. Actions applied to defects . . . 266
5.7.4. Defect disposition . . . 266
5.8. Sample exam questions . . . 267

Chapter 6. Tools Support for Testing . . . 277

6.1. Types of test tools . . . 277
6.1.1. Test tool classification . . . 278
6.1.2. Tools supporting test management . . . 278
6.1.3. Tools supporting requirement management . . . 279
6.1.4. Tools supporting static tests . . . 279
6.1.5. Modeling tools . . . 280
6.1.6. Tools supporting test design and test data creation . . . 280
6.1.7. Tools supporting test execution . . . 281
6.1.8. Tools supporting test environment management . . . 281
6.1.9. Tools supporting test data comparison . . . 282
6.1.10. Tools supporting test coverage measurement . . . 282
6.1.11. Other test supporting tools . . . 282
6.2. Assumptions and limitations of test tools . . . 283
6.2.1. Advantages and risks of the tools . . . 283
6.2.2. Specific considerations for some tools . . . 285
6.3. Selecting and introducing tools in an organization . . . 289
6.3.1. Main principles . . . 289

6.3.2. Tool selection process 290
6.3.3. Test tool implementation 293
6.3.4. To build or to buy test tools? 294
6.4. Sample exam questions 295

Chapter 7. Mock Exam 299

Chapter 8. Templates and Models 313

8.1. Master test plan 313
8.2. Test plan 315
8.2.1. Test plan as per IEEE 829-1998 315
8.2.2. Test plan as per IEEE 829-2008 315
8.3. Test design document 317
8.3.1. Test design specifications as per IEEE 829-1998 317
8.3.2. Test design document as per IEEE 829-2008 317
8.4. Test case 318
8.4.1. Test case document as per IEEE 829-1998 318
8.4.2. Test case document as per IEEE 829-2008 318
8.5. Test procedure 319
8.5.1. Test procedure document as per IEEE 829-1998 319
8.5.2. Test procedure document as per IEEE 829-2008 319
8.6. Test log 320
8.6.1. Test log as per IEEE 829-1998 320
8.6.2. Test log as per IEEE 829-2008 320
8.7. Defect report 320
8.7.1. Defect report as per IEEE 829-1998 320
8.7.2. Defect report as per IEEE 829-2008 321
8.8. Test report 322
8.8.1. Test report as per IEEE 829-1998 322
8.8.2. Interim test report as per IEEE 829-2008 322
8.8.3. Level test report as per IEEE 829-2008 323
8.8.4. Master test report as per IEEE 829-2008 323

Chapter 9. Answers to the Questions 325

9.1. Answers to the end-of-chapter questions 325
9.2. Correct answers to the sample paper questions 327

References 329

Index 333

Preface

"My eldest brother sees the spirit of sickness and removes it before it takes shape, so his name does not get out of the house. My elder brother cures sickness when it is still extremely minute, so his name does not get out of the neighborhood. As for me, I puncture veins, prescribe potions, and massage skin, so from time to time my name gets out and is heard among the lords".

– Sun Tzu, The Art of War

I often turn to the above quote, replacing "sickness" by "defects", and applying it to software instead of humans. I have seen few eldest brothers, a number of elder ones, and am perhaps in the last category of practitioners.

Why this book?

As we know, software testing is increasingly important in the industry, reflecting the increasing importance of software quality in today's world. Since 2011, when the first edition of this book was published, there have been evolutions in the software industry that have impacted software testing. This new – revised – edition will help testers adopt more up-to-date fundamental practices.

Due to the lack of formal and recognized training in software testing, a group of specialist consultants gathered together in 2002 and founded the International Software Testing Qualifications Board (ISTQB). They defined the minimal set of methodological and technical knowledge that testers should know depending on their experience. This was gathered into what is called a syllabus. The foundation level syllabus was reviewed in 2023 and has been the basis of an international certification scheme that has already been obtained by more than 1,000,000 testers

worldwide. This book can serve as reference material for testers preparing the ISTQB foundation level exam, and for any beginner testers. It references the 2023 version of the ISTQB Certified Tester Foundation Level syllabus.

This book follows the order and chapters of the syllabus, which should help you to successfully complete the certification exam. It is a one-stop reference book offering you:

– more detailed explanations than those found in the ISTQB syllabus;

– definitions of the terms (i.e. the Glossary) used in the certification exams;

– practice questions similar to those encountered during the certification exam;

– a sample exam.

For testers who want to acquire a good understanding of software and system tests, this book provides the fundamental principles as described by the ISTQB and recognized experts.

This book provides answers and areas of discussion to enable test leaders and managers to:

– improve their understanding of testing;

– have an overview of process improvement linked to software testing;

– increase the efficiency of their software development and tests.

Throughout this book, you will find learning objectives (denoted as FL-…) that represent the ISTQB foundation level syllabus learning objectives. These are the topics that certification candidates should know and that are examined in certification exams.

Prerequisite

Software testing does not require specific prerequisites, but a basic understanding of data processing and software will allow you to better understand software testing.

The reader with software development knowledge, whatever the programming language, will understand certain aspects faster, but a simple practice as a user should be enough to understand this book.

ISTQB and national boards

The ISTQB is a not-for-profit international association grouping national software testing boards covering over 50 countries. These national boards are made up of software testing specialists, consultants and experts, and together they define the syllabi and examination directives for system and software testers.

A number of prominent authors of software testing books participated in the creation of the initial syllabi, ensuring that they reflect what a tester should know depending on their level of experience (foundation, advanced, expert) and their objectives (test management, functional testing and test techniques, specialization in software security or performance testing, etc.).

Glossary, syllabus and business outcomes

The ISTQB is aware of the broad diversity of terms used and the associated diversity of interpretation of these terms depending on the customers, countries and organizations. A common glossary of software testing terms has been set up and national boards provide translation of these terms in national languages to promote better understanding of the terms and the associated concepts. This becomes increasingly important in a context of international cooperation and offshore sub-contracting.

The syllabi define the basis of the certification exams; they also help to define the scope of training and are applicable at three levels of experience: foundation level, advanced level and expert level. This book focuses on the foundation level.

The foundation level, made up of a single module, is detailed in the following chapters.

Expected business outcomes, as stated by the ISTQB, are as follows for foundation level testers:

– understand what testing is and why it is beneficial;

– understand the fundamental concepts of software testing;

– identify the test approach and activities to be implemented depending on the context of testing;

– assess and improve the quality of documentation;

– increase the effectiveness and efficiency of testing;

– align the test process with the software development life cycle;

– understand test management principles;

– write and communicate clear and understandable defect reports;

– understand the factors that influence the priorities and efforts related to testing;

– work as part of a cross-functional team;

– know the risks and benefits related to test automation;

– identify the essential skills required for testing;

– understand the impact of risk on testing;

– effectively report on test progress and quality.

As we can see, the work of a tester impacts many different aspects in software development, from evaluating the quality of input documentation (specifications, requirements, user stories, etc.) to reporting on progress and risks, to test automation and interacting with the development teams to understand what to test and explain what defects are identified.

ISTQB certification

The ISTQB proposes software tester certifications, which are recognized as equivalent by all ISTQB member boards throughout the world. The level of difficulty of the questions and the exams are based on identical criteria (defined in the syllabi) and terminology (defined in the Glossary).

The certification exams proposed by the ISTQB enable the candidates to validate their knowledge, and assure employers or potential customers of a minimum level of knowledge from their testers, whatever their origin. Training providers deliver courses to help participants succeed in the certification exams, however much of the training involves brain cramming sessions and does not ensure that the participant has the required level of autonomy to succeed in the profession. This book attempts to identify the necessary skills, as well as provide the reader with a concentrate of more than 40 years of practice in the field of software quality and testing.

The ISTQB certifications are recognized as equivalent throughout the whole world, enabling international cross-recognition.

Key for understanding the content

To be used efficiently, this book has the following characteristics:

FL-xxx: text that starts with FL-xxx is a reminder of the learning objectives present in the ISTQB foundation level syllabus for certified testers. Those objectives are expanded in the paragraphs following this tag.

The titles of the chapters correspond to those of the ISTQB foundation level syllabus, version 2011. This is also often the case for the section heads; the syllabus reference is provided in the form (FLx.y), where x.y stands for the chapter and section head of the ISTQB foundation level syllabus.

A synopsis closes each of the chapters, summarizing the aspects covered and identifying the terms in the glossary that should be known for the certification exam. Sample exam questions are also provided at the end of each chapter. These questions were developed by applying the same criteria as for the creation of real exam questions.

The sample questions provided in Chapters 1–6 have been reproduced with the kind permission of © Bernard Homès 2011.

March 2024

Glossary

The definitions listed below have been extracted from the International Software Testing Qualifications Board (ISTQB) *Standard Glossary of Terms used in Software Testing*. Only the terms used for the Foundation Level certification exams are mentioned, so as not to drown the reader in terms that are used at other levels or in other syllabi.

Acceptance criteria: The criteria that a component or system must satisfy in order to be accepted by a user, customer or other authorized entity (from ISO 24765).

Acceptance test-driven development: A collaboration-based test-first approach that defines acceptance tests in the stakeholders' domain language. **Abbreviation:** ATDD.

Acceptance testing: Formal testing with respect to user needs, requirements and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entities to determine whether or not to accept the system. See also *user acceptance testing*.

ACM: (Association for Computer Machinery) professional and scientific association for the development of information technology.

Alpha testing: Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization. Alpha testing is often employed as a form of internal acceptance testing.

Anomaly: A condition that deviates from expectation (from ISO 24765).

Attack: Directed and focused attempt to evaluate the quality, and especially the reliability, of a test object by attempting to force specific failures to occur.

Beta testing: Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes. Beta testing is often employed as a form of external acceptance testing in order to acquire feedback from the market.

Black-box technique: See also *black-box testing.*

Black-box test technique: A test technique based on an analysis of the specification of a component or system. **Synonyms:** black-box test design technique, specification-based test technique.

Black-box testing: Testing, either functional or nonfunctional, based on an analysis of the specification of the component or system. **Synonym:** specification-based testing.

Boundary value analysis: A black-box test technique in which test cases are designed based on boundary values. See also *boundary values*.

Branch coverage: The coverage of branches in a control flow graph (percentage of branches that have been exercised by a test suite). One hundred percent branch coverage implies both 100% decision coverage and 100% statement coverage.

Bug: See also *defect*.

Checklist-based testing: An experience-based test technique in which test cases are designed to exercise the items of a checklist.

Code coverage: An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, for example, statement coverage, decision coverage or condition coverage.

Collaboration-based test approach: An approach to testing that focuses on defect avoidance by collaborating among stakeholders.

Commercial off-the-shelf software (COTS): See also *off-the-shelf software*.

Compiler: A software tool that translates programs expressed in a high-order language into their machine language equivalents.

Complexity: The degree to which a component or system has a design and/or internal structure that is difficult to understand, maintain and verify. See also *cyclomatic complexity*.

Component integration testing: The testing executed to identify defects in the interfaces and interactions between integrated components. **Synonyms:** module integration testing, unit integration testing.

Component testing: A test level that focuses on individual hardware or software components. **Synonyms:** module testing, unit testing.

Configuration control: An element of configuration management, consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

Configuration item: An aggregation of hardware, software or both, that is designated for configuration management and treated as a single entity in the configuration management process.

Configuration management: A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

Confirmation testing: A type of change-related testing performed after fixing a defect to confirm that a failure caused by that defect does not reoccur. **Synonym:** re-testing.

Control flow: An abstract representation of all possible sequences of events (paths) in the execution through a component or system.

Coverage: The degree to which specified coverage items are exercised by a test suite, expressed as a percentage**. Synonym:** test coverage.

Coverage item: An attribute or combination of attributes derived from one or more test conditions by using a test technique. See also *coverage criteria*.

Coverage measurement tool: See also *coverage tool*.

Coverage tool: A tool that provides objective measures of what structural elements, for example, statements, branches, have been exercised by the test suite.

Cyclomatic complexity: The number of independent paths through a program. Cyclomatic complexity is defined as: L – N + 2P, where:

– L = the number of edges/links in a graph;

– N = the number of nodes in a graph;

– P = the number of disconnected parts of the graph (e.g. a calling graph and a subroutine).

Data-driven testing: A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools. See also *keyword-driven testing*.

Data flow: An abstract representation of the sequence and possible changes of the state of data objects, where the state of an object can be creation, usage or destruction.

Debugging: The process of finding, analyzing and removing the causes of failures in a component or system.

Debugging tool: A tool used by programmers to reproduce failures, investigate the state of programs and find the corresponding defect. Debuggers enable programmers to execute programs step-by-step, halt a program at any program statement and set and examine program variables.

Decision coverage: The percentage of decision outcomes that have been exercised by a test suite. One hundred percent decision coverage implies both 100% branch coverage and 100% statement coverage.

Decision table testing: A black-box test technique in which test cases are designed to exercise the combinations of conditions inputs and/or stimuli (causes) shown in a decision table.

Defect: An imperfection or deficiency in a work product, which can cause the component or system to fail to perform its required function, for example, an incorrect statement or data definition (from ISO 24765). A defect, if encountered during execution, may cause a failure of the component or system. **Synonyms:** bug, fault.

Defect density: The number of defects identified in a component or system divided by the size of the component or system (expressed in standard measurement terms, for example, lines-of-code, number of classes or function points).

Defect management: The process of recognizing, recording, classifying, investigating, resolving and disposing of defects. It involves recording defects, classifying them and identifying the impact.

Defect management tool: See also *incident management tool*.

Defect report: Documentation of the occurrence, nature and status of a defect. **Synonym:** bug report.

Driver: A software component or test tool that replaces a component that takes care of the control and/or the calling of a component or system.

Dynamic analysis tool: A tool that provides run-time information on the state of the software code. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic, and monitor the allocation, use and de-allocation of memory and highlight memory leaks.

Dynamic testing: Testing that involves the execution of the test item/software of a component or system (from ISO 29119-1). See also *static testing*.

Entry criteria: The set of generic and specific conditions that permit a process to proceed with a defined task (from Gilb and Graham), for example, test phase. The purpose of entry criteria is to prevent a task that would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria from starting. See also *exit criteria*.

Equivalence partitioning: A black-box test technique in which test conditions are equivalence partitions exercised by one representative member of each partition (from ISO 29119-1). **Synonym:** partition testing.

Error: A human action that produces an incorrect result (from ISO 24765). **Synonym:** mistake.

Error guessing: A test design technique in which tests are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes, in order to anticipate the defects that may be present in the component or system under test as a result of errors made, and design tests specifically to expose them (from ISO 29119-1).

Exhaustive testing: A test approach in which the test suite comprises all combinations of input values and preconditions.

Exit criteria: The set of generic and specific conditions, agreed upon with the stakeholders, that permit a process to be officially completed. The purpose of exit

criteria is to prevent a task from being considered completed when there are still outstanding parts of the task that have not been finished. Exit criteria are used by testing to report against and plan when to stop testing (after Gilb and Graham). **Synonyms:** test completion criteria, completion criteria. See also *entry criteria*.

Experience-based test technique: A test technique based on the tester's experience, knowledge and intuition. **Synonyms:** experience-based test design technique, experience-based technique.

Exploratory testing: An approach to testing in which the testers dynamically design and execute tests based on their knowledge, exploration of the test item and the results of previous tests. This is used to design new and better tests (from ISO 29119-1). See also *test charter*.

Failure: An event in which a component or system does not perform a required function within specified limits (from ISO 24765). Actual deviation of the component or system from its expected delivery, service or result (according to Fenton). The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered [EUR 00].

Failure rate: The ratio of the number of failures of a given category to a given unit of measure, for example, failures per unit of time, failures per number of transactions and failures per number of computer runs.

Fault attack: See also *attack*.

Field testing: See also *beta testing*.

Finite state testing: See also *state transition testing*.

Formal review: A type of review that follows a defined process with a formally documented output, for example, inspection (from ISO 20246).

Functional requirement: A requirement that specifies a function that a component or system must perform.

Functional testing: Testing performed to evaluate if a component or system satisfies functional requirements (from ISO 24765). See also *black-box testing*.

Horizontal traceability: The tracing of requirements for a test level through the layers of test documentation (e.g. test plan, test design specification, test case specification and test procedure specification).

IEEE: Institute for Electrical and Electronic Engineers, a professional, not-for-profit association for the advancement of technology, based on electrical and electronic technologies. This association is active in the design of standards. There is a French chapter on this association, which provides publications that are useful for software testers.

Impact analysis: The assessment of change to the layers of development documentation, test documentation and components, in order to implement a given change to specified requirements.

Incident: Any event occurring during testing which requires investigation.

Incident management tool: A tool that facilitates the recording and status tracking of incidents found during testing. They often have workflow-oriented facilities to track and control the allocation, correction and re-testing of incidents and provide reporting facilities. See also *defect management tool*.

Incident report: A document reporting on any event that occurs during the testing which requires investigation.

Incremental development model: A development life cycle where a project is broken into a series of increments, each of which delivers a portion of the functionality in the overall project requirements. The requirements are prioritized and delivered in priority order in the appropriate increment. In some (but not all) versions of this life cycle model, each sub-project follows a "mini V-model" with its own design, coding and testing phases.

Independence of testing: Separation of responsibilities, which encourages the accomplishment of objective testing.

Informal review: A type of review that does not follow a defined process and has no formally documented output.

Inspection: A type of formal review that relies on visual examination of documents to detect defects, for example, violations of development standards and non-conformance to higher-level documentation, and uses defined team roles and measurements to identify defects in a work product and improve the review and software development processes. The most formal review technique and, therefore, always based on a documented procedure (from ISO 20246). See also *peer review*.

Intake test: A special instance of a smoke test to decide whether the component or system is ready for detailed and further testing. An intake test is typically carried out at the start of the test execution phase. See also *smoke test*.

Integration: The process of combining components or systems into larger assemblies.

Integration testing*:* Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also *component integration testing*, *system integration testing*.

Interoperability testing: The process of testing to determine the interoperability of a software product.

ISTQB: International Software Testing Qualifications Board, a nonprofit association developing international certification for software testers.

Keyword-driven testing: A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test. See also *data-driven testing*.

Maintainability testing: The process of testing to determine the maintainability of a software product.

Maintenance testing: Testing the changes to an operational system or the impact of a changed environment on an operational system.

Master test plan: See also *project test plan*.

Metric: A measurement scale and the method used for measurement.

Mistake: See also *error*.

Modeling tool: A tool that supports the validation of models of the software or system.

Moderator: The leader and main person responsible for an inspection or other review process.

Non-functional testing: Testing performed to evaluate whether a component or system complies with nonfunctional requirements.

N-switch coverage: The percentage of sequences of N+1 transitions that have been exercised by a test suite.

N-switch testing: A form of state transition testing in which test cases are designed to execute all valid sequences of N+1 transitions (Chow). See also *state transition testing*.

Off-the-shelf software: A software product that is developed for the general market, that is, for a large number of customers, and that is delivered to many customers in identical format.

Oracle: See also *test oracle*.

Peer review: See also *technical review*.

Performance testing: The process of testing to determine the performance of a software product.

Performance testing tool: A tool to support performance testing, that usually has two main facilities: load generation and test transaction measurement. Load generation can simulate either multiple users or high volumes of input data. During execution, response time measurements are taken from selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

Portability testing: The process of testing to determine the portability of a software product.

Probe effect: The effect on the component or system when it is being measured, for example, by a performance testing tool or monitor. For example, performance may be slightly worse when performance testing tools are being used.

Product risk: A risk impacting the quality of a product and directly related to the test object. See also *risk*.

Project risk: A risk related to management and control of the (test) project, for example, lack of staffing, strict deadlines, changing requirements, etc., that impacts project success. See also *risk*.

Project test plan: A test plan that typically addresses multiple test levels. See also *master test plan*.

Quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations (from IREB).

Quality assurance: Activities focused on providing confidence that quality requirements will be fulfilled. **Abbreviation:** QA (from ISO 24765). See also *quality management*.

RAD: Rapid Application Development, a software development model.

Regression testing: A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software. It is performed when the software or its environment is changed.

Reliability testing: The process of testing to determine the reliability of a software product.

Requirement: A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document.

Requirement management tool: A tool that supports the recording of requirements, attributes of requirements (e.g. priority, knowledge responsible), and annotation, and facilitates traceability through layers of requirements and requirement change management. Some requirement management tools also provide facilities for static analysis, such as consistency checking and violations to pre-defined requirement rules.

Re-testing: Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

Review: A type of static testing in which a work product or process is evaluated by one or more individuals to detect defects or provide improvements. Examples include management review, informal review, technical review, inspection and walk-through.

Review tool: A tool that provides support to the review process. Typical features include review planning, tracking support, communication support, collaborative reviews and a repository for collecting and reporting of metrics.

Reviewer: The person involved in the review who identifies and describes anomalies in the product or project under review. Reviewers can be chosen to represent different viewpoints and roles in the review process.

Risk: A factor that could result in future negative consequences; usually expressed as impact and likelihood. See also *product risk*, *project risk*.

Risk analysis: The overall process of risk identification and risk assessment.

Risk assessment: The process to examine identified risks and determine the risk level.

Risk-based testing: Testing in which the management, selection, prioritization and use of testing activities and resources are based on corresponding risk types and risk levels. This approach is used to reduce the level of product risks and inform stakeholders about their status, starting in the initial stages of a project (from ISO 29119-1).

Risk control: The overall process of risk mitigation and risk monitoring.

Risk identification: The process of finding, recognizing and describing risks. (from ISO 31000).

Risk level: The measure of a risk defined by risk impact and risk likelihood. **Synonym:** risk exposure.

Risk management: The process for handling risks (from ISO 24765).

Risk mitigation: The process through which decisions are reached and protective measures are implemented to reduce risks or maintain them at specified levels.

Risk monitoring: The activity that checks and reports the status of known risks to stakeholders.

Robustness testing: Testing to determine the robustness of the software product.

Root cause: A source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed (from CMMI).

SBTM: Session-based test management, an ad hoc and exploratory test management technique, based on fixed length sessions (from 30 to 120 minutes), during which testers explore a part of the software application.

Scribe: The person who has to record each defect mentioned and any suggestions for improvement during a review meeting, on a logging form. The scribe has to ensure that the logging form is readable and understandable.

Scripting language: A programming language in which executable test scripts are written, used by a test execution tool (e.g. a capture/replay tool).

Security testing: Testing to determine the security of the software product.

Shift left: An approach to perform testing and quality assurance activities as early as possible in the software development life cycle.

Site acceptance testing: Acceptance testing by users/customers on site, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes, normally including hardware as well as software.

SLA: Service-level agreement, service agreement between a supplier and their client, defining the level of service a customer can expect from the provider.

Smoke test: A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertain that the most crucial functions of a program work, but not bothering with finer details. A daily build and smoke test is among industry best practices.

State transition: A transition between two states of a component or system.

State transition testing: A black-box test design technique in which test cases are designed to exercise elements of a state transition model, and execute valid and invalid state transitions. **Synonym:** finite state testing. See also *N-switch testing*.

Statement coverage: The percentage of executable statements that have been exercised by a test suite.

Static analysis: The process of evaluating a component or system without executing it, based on its form, structure, content or documentation, for example, requirements or code, carried out without execution of these software artifacts (from ISO 24765).

Static code analyzer: A tool that carries out static code analysis. The tool checks the source code for certain properties, such as conformance to coding standards, quality metrics or data flow anomalies.

Static testing: Testing of a component or system at the specification or implementation level without execution of that software, for example, reviews or static code analysis. See also *dynamic testing*.

Stress testing: A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified workloads, or with reduced availability of resources, such as access to memory or servers. See also *performance testing*, *load testing*.

Stress testing tool: A tool that supports stress testing.